Created: December 7, 1998
Modified: November 9, 1998

Gemini Telescope Control System Report

The TCS Time System - ICD21

Chris Mayer, Dave Terrett & Pat Wallace

tcs_cjm_034.fm/09

This report is the User and Programmers manual for the TCS Time System. It describes the support provided by the TCS for different time systems both for itself and for other Gemini EPICS systems.

1.0 Introduction

1.1 Intended Readership

This document is aimed at *all* Gemini work packages that are developing EPICS/ VxWorks systems and require access to time in a variety of formats.

It should be emphasised that ***all*** *access to time by* ***any*** *Gemini EPICS system should be through the facilities described in this document. Systems should* ***not*** *access time by making low level calls to the Bancomm card e.g. through the bc635_read call [1].*

1.2 Scope

The software and facilities described in this document are mainly applicable to those Gemini work packages that are developing EPICS based VxWorks systems. The software currently requires the facilities of both the EPICS sequencer and the EPICS database.

The software only addresses the problem of accessing time from C code. ICD 9 [1] describes the facilities available to access time via EPICS records.

The software and facilities described here will be referred to as the TCS Time System.

1.3 Acronyms and abbreviations

DHS	Data Handling System
EPICS	Experimental Physics and Industrial Control System

GMST	Greenwich Mean Sidereal Time
IOC	Input Output Controller
LAST	Local Apparent Sidereal Time
RTC	Real Time Clock
TAI	International Atomic Time
TCS	Telescope Control System
TDB	Barycentric Dynamical Time
TT	Terrestrial Time
UTC	Coordinated Universal Time
UT1	Universal Time

1.4 Definitions

Raw time - a double precision number containing the number of SI seconds since 1970 January 1.0 International Atomic Time

1.5 References

[1] *ICD 9 - EPICS Time Bus Driver,* P. B. Taylor

[2] *SPE-C-G0009 - Software Programming Standards*, P. McGehee & S. Wampler

[3] *TCS/PTW/6 - Time*, P.T. Wallace.

1.6 History

v 5.0 - 3 June 1997 - update reference section and required database records

v 6.0 - 17 June 1997 - added extra info and corrected ambiguities following input from PCS work package

v7.0 - 4 June 1998 - fix description of timeOffset routine

v8.0 - modify description of timePrint to reflect implementation of release V1-1

v9.0 - added sections on use of timelib off-line and how timelib initialises

2.0 Overview

The requirements for the TCS Time System initially stemmed from the needs and duties of the TCS Work Package. These were

- The TCS requires accurate time on a number of different timescales
- The TCS is the time bus master and normally has access to accurate time via a Bancomm 637 card and GPS receiver
- The TCS must be able to run when a Bancomm card is not available

Although these are requirements on the TCS, all Gemini EPICS systems also require access to accurate time on a range of timescales. Since the details of how to convert time as read from the Bancomm hardware to any particular timescale are complex and subtle it was decided to extend the facilities provided by the TCS as time bus master to provide an enhanced Time System.

Some of the facilities provided by the TCS Time System are

1. Access to “raw time” from the best available time source
2. Ability to run a subsystem or the whole observatory on a time offset from the present
3. Ability to “simulate” time when the GPS or IRIG-B signals are not available or when no Bancomm hardware is present.
4. Automatic connection to the TCS when it is running and disconnection when it is shutdown
5. Access to accurate time as TAI, UTC, UT1, TT, TDB, GMST or LAST
6. Maintenance of a health record for the time system for inclusion in the overall system health as well as logging of messages to a DHS log record

3.0 User Manual

This section of the document comprises the User Manual. Only those aspects of the Time System that need to be understood to use the system are described here. Details of each component of the Time system can be found in Section 5.0 .

3.1 timeClockInit

The first call that a system must make is to timeClockInit (Section 5.3). This call must be made before the EPICS Timestamp system is initialised and before IocInit.

timeClockInit takes five parameters. The first indicates whether the system is the time bus master (= 1) or not (= 0). Only the TCS is expected to call timeClockInit with the first parameter set to 1. All other systems should set the first parameter to 0. The second parameter indicates whether the time system is to be run for “real” (= 0) or it is to be “simulated” (= 1). In normal operational use all systems will set this second parameter to 0.

Time should be simulated i.e. this second parameter should be set to 1 under the following circumstances

1. When no Bancomm time card is available. In this case time will be derived from the CPU RTC.
2. When the TCS is not available and/or when there is no physical time bus cable. This situation is likely to be the case for most work package developers before their systems are integrated at Mauna Kea or Cerro Pachon. In this case time will be derived from the RTC on the Bancomm card.

   System developers are free to run their systems with this second parameter set to 0 without the time bus or the TCS. Under these circumstances the <sys>:TIME:health record will show Warning or Bad depending on how far in error the time is.

The remaining three parameters are passed directly to the equivalent parameters in the routine BCconfigure as documented in [1]. These parameters are the frequency of periodic interrupts as generated by the Bancomm card, the ratio of the periodic interrupts to the system clock tick rate and an offset in microseconds that can be used to compensate for delays in the time reference input.

3.2 Database records

Correct operation of the Time system requires that the IOC load three Time system specific EPICS records. These are currently <sys>TIME:health <sys>TIME:intSimulate and <sys>logrecord. The Time System uses the first of these to provide information on its health and the second is a record for the DHS to monitor. The intSimulate record indicates whether time is being simulated or not. The token <sys> should be replaced with whatever the standard prefix is for the subsystem in question as described in [2] e.g. for the PCS it would be m1: (Note the colon that is part of the token). It is the responsibility of the system developer to make sure records with these names are part of the database they load.

N.B. In tests so far these records have been implemented as an ao (health), a stringin (logrecord) and a bo (intSimulate). (see Section 4.3 on page 10)

3.3 timeSeq

Monitoring of the time system and updating the health and log record is carried out by a sequence program called timeSeq. This should be started as follows

seq &timeSeq, “sys=<sys>”

Where the token <sys> is replaced by the standard prefix for the system in question and must correspond to the name used to prefix the health and log records. (for further details see Section 4.1.1 on page 9)

3.4 Startup script

An example startup script is shown below. Those commands important to the operation of the Time System have been highlighted

cd to application directory

cd “/home/vxWorks/tcs1/cjm/proto/rel”

Load EPICS core and record stuff

ld < epics/base/bin/mv167/iocCore

ld < epics/base/bin/mv167/drvSup

ld < epics/base/bin/mv167/recSup

ld < epics/base/bin/mv167/devSup

If using state programs, load the sequencer

ld < epics/base/bin/mv167/seq

Load any application records, device support or drivers here,

```
# or code for subroutine records
ld < ../../slalib/rel/bin/mv167/slalib
ld < ../../timelib/rel/bin/mv167/timelib
# Load sequences here
ld < ../../timelib/rel/bin/mv167/timeSeq
# Configure the time system
# Master ?, Simulate? interrupt frequency, intPerTick, timeOffset
timeClockInit (0, 0, 60, 1, 0)
# Configure the Timestamp system, 1=Master IOC
# Master?, synch rate, clock rate, master port, slave port, timeout (ms), type
# If value is set to 0 then the default compilation value is used
# Type = 1 permanently inhibits the use of the event system
TSconfigure 1,0,0,0,0,250,1
# Load one or more databases here
dbLoad "./data/defualt.dctsdr"
dbLoadRecords  "./data//xxx.db"
# access security initialization
# asSetFilename("src/geminiTcs.acf")
# This turns off logging
# iocLogDisable=1
iocInit "./data/resource.def"
# Start sequences
seq &timeSeq, "sys=m1:"
```

The important points to note about this startup script are

1. slalib must be loaded prior to timelib as timelib makes calls to routines in slalib. The Project Office has not specified a standard location from which to load these libraries so application developers are free to chose. The example here loads them from their own UAE directory tree but this is not compulsory.
2. timeClockInit must be called prior to configuring the EPICS Time Stamp system and before EPICS driver initialisation. Further details of this function can be found in Section 5.3 . The parameters given here are appropriate to a system that is not the time bus master but that intends to use its Bancomm hardware locked to an external time signal e.g. IRIG-B. The interrupt frequency is set to 60Hz as is the system clock rate.
3. The database xxx.database must contain the records referred to in Section 3.2
4. resource.def should contain the IP address of an NTP time server. This is used to set the CPU RTC as well as to periodically cross check the time on the Bancomm card.
5. The time sequence is started on the assumption that the three records referred to in Section 3.2 and loaded by xxx.database are called m1:TIME:health, m1:TIME:intSimulate and m1:logrecord.

3.5 Calling the Time System from Application code

The first thing an application developer should remember is that no-one except the TCS should call the Bancomm driver directly! All access to time should be through one of the library calls detailed in Section 5.0 .

All the details of handling the initialisation of the time system are handled by the sequence program timeSeq (Section 3.3). Once this is loaded and started, a user only need make calls to the timeNowX and timeThenX routines (Section 5.10 to Section 5.26).

The lowest level call an application developer should make is to the timeNow function. For example all inter IOC traffic should use times as returned by this function. The format of this time is identical to that of the bc635_read call i.e. a double precision number containing the number of SI seconds since 1970 January 1.0 TAI. This is known as a "raw time"

If one system wants to command another to be at a certain position at a certain time then the commanding system will send the required time as a double in this format. The commanded system will call timeNow to get the current time in the same format and difference the two numbers to find the number of seconds left before it must be in the position required.

Programmers are strongly discouraged from manipulating raw times themselves, apart from simple transformations such as adding or subtracting a short interval. Conversion into date and time format, or into different timescales should only be carried out through the time library functions described in Section 5.0 on page 15.

3.6 Running the observatory or subsystem at a time offset from the present

If it is required to run the whole observatory at a time offset from the present then the TCS and all other systems must have been started with the second parameter to timeClockInit set to 0. Simply writing into TCS:biass will then cause all systems to adjust to the same new offset time

(The name of this record and how it is written to will change. A TCS CAD record will be provided in later versions of the TCS.)

If a system has been started with time being simulated then no connections are made to the TCS so changing the value of the TCS:biass record will have no affect on it. If a system wishes to run on a time offset from the present under these circumstances then it must provide an engineering function that makes a call to the timeOffset routine. (Section 5.16) It is essential that this engineering function is not invoked when time is being derived from the TCS. If it were, then a system would end up running on a time offset from the rest of the observatory.

3.7 Using timelib on a Unix host

Although timelib was initially written to provide time services for VxWorks IOCs, it can be used essentially unchanged on a Unix host. The only difference is the manner in

which it must be initialised since the Unix host does not have access to the time bus hardware.

For host based applications timelib is initialised by a call to timeOffline (Section 5.15 on page 22) rather than a call to timeInit. If timeInit is called on a non-VxWorks system then an error is generated. Similarly, calling timeOffLine on a VxWorks system will also generate an error.

Due to the limitations of the underlying operating system, timelib cannot guarantee a time resolution of better than 1 second for host based applications although higher resolution clocks are used if available.

4.0 Programmer's Manual

This section provides a functional description of the main components of the TCS Time System, in particular it describes the operation of the sequence program that initialises and monitors the operation of the time system on each IOC on which it is running

4.1 Sequence program

The operation of the sequence program is best understood by reference to the state transition diagram shown in Figure 1 .

On startup the sequence enters the state "Connecting". At this point it is checking that all the connections that it needs to the TCS are available i.e. that the TCS is up and running somewhere on the network. There are three possible exits from this state

1. The time system is being simulated. In this case the sequence immediately transitions to the state "Simulating". Currently this state does nothing. In particular there is no way of switching the system dynamically to run in non-simulate mode. This can only be achieved by a reboot.
2. Not all channels are connected. In this situation the system will transition to state "Wait Connect". It will stay in this state until the channels become available. This will happen when the TCS is booted. at which point it will move to state "Check State"
3. All channels connected. In this case the system transitions immediately to state "Check State"

The state "Check state" is used to control access to the TCS whilst it is starting up and initialising its internal data. EPICS will report that all channels to the TCS are available once the TCS database is loaded. At this point however the TCS records that the time system requires will not have been initialised to sensible values. This will occur once the TCS has started its main pointing loops. As soon as the TCS has performed its own internal initialisations it will set the record TCS:state to the value Running. This signals to the time system that valid data is available and the sequence moves to the state "Monitoring"

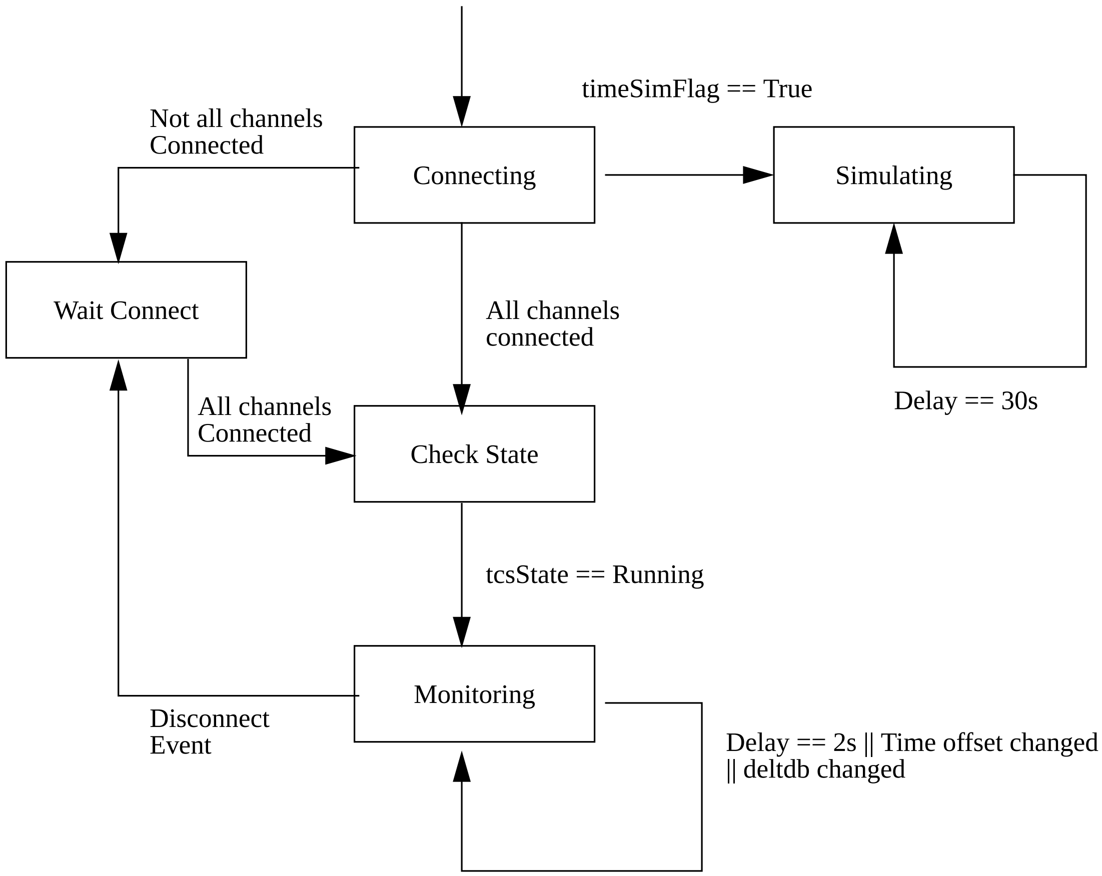

FIGURE 1. State Transition diagram for timeSeq

The monitoring state has two main functions

1. To update the local system's copies of the TCS Time System variables
2. To maintain a record of the time system's health

The first function ensures that the local time system can provide time as accurately as possible on a variety of timescales without having to duplicate the calculations being done by the TCS.

The second function relieves a systems application code from having to handle error returns from the Bancomm hardware. For example, The Bancomm card sets error bits when its estimate of the time error is greater than 5 µsecs . Unfortunately there is no way of telling how much worse than 5 µsecs the error is. The sequence program monitors the error bits and as soon as they get set it sets the time system health to Warning. It then checks the time against an NTP server and if the error is greater than 2 seconds then sets the health to Bad.

The rather crude error band of 2 seconds is to ensure the health does not get set Bad during leap second updates. In practice tests have shown than the Bancomm and NTP servers never get more than 30ms out of step. Later releases of the TCS Time system may incorporate more sophisticated error checking.

4.1.1 Connecting to an alternate TCS

The timeSeq program is normally started as shown in Section 3.3 . By default this will try and connect to a TCS database that is prefixed tcs: For testing purposes it may be convenient to connect to an alternate TCS database if one is running. This can be done by over riding the default "top" macro used by timeSeq. For example to connect to a TCS database prefixed tc1: you should start timeSeq as follows

```
seq &timeSeq "sys=m1:, top=tc1:"
```

4.2 Initialisation

The initialisation of the TCS time system depends on the hardware that is available and the requirements of the user. It is based on the assumption that the best source of time should be used if possible and that these are (in decreasing order of accuracy)

1. The GPS signal to the Bancomm 637 in the TCS crate and by extension the IRIG-B signal sent to the other IOCs
2. The RTC on the Bancomm cards
3. The RTC on the mv167

The initialisation is performed by the routines timeClockInit and timeInit and the table below summarizes the functions that are performed both when simulating and when not. In this context, "simulating" means not using the Bancomm 637 and its associated IRIG-B signal.

TABLE 1. Initialisation functions

	simulated	Not simulated
timeClockInit	Set system clock rate directly	Call BCconfigure so that system clock rate is set by interrupts from Bancomm
timeInit	Read NTP and set CPU RTC to UTC If Bancomm present sets its RTC	Read NTP and set CPU RTC to UTC

The points to note are

- The CPU RTC is always set whether simulating or not
- The Bancomm RTC is only set if simulating.

  The assumption here is that if not simulating then a valid time signal is already available either direct from the GPS receiver in the case of the TCS or via the time bus in the case of other IOCs. Since this is the highest precision signal available it makes no sense to attempt to over ride it.

  A consequence of this behaviour is that if a crate is rebooted by cycling the power or pressing the reset button the Bancomm time must be reset by either

1. Having the time bus connected

   OR

2. Running the time system in simulation mode

   Failure to do either of these will result in invalid times as the Bancomm starts counting time from zero after a bus reset.

4.3 Database records

The TCS Time System relies on the following records being loaded by the TCS

TABLE 2. Records loaded by the TCS

Record	Type	Description
state	mbbi	Booting, Initialising or Running
biass	ai	Time offset (days)
deltdb	ai	TDB - TT (days)
tlongm	ai	Site mean east longitude
tlatm	ai	Site mean geodetic latitude
xpm	ai	X component of polar motion (arcsec)
ypm	ai	Y component of polar motion (arcsec)
ttmtai	ai	TT - TAI (secs)
djmls	ai	MJD following next leap second
delat	ai	TT - TAI (secs)
delut	ai	UT1 - UTC (secs)

In addition each system that uses the TCS Time system must provide the following records

TABLE 3. Records provided by each system

Name	Type	Description
<sys>TIME:health	ao	0 = Good, 100 = Warning or 200 = Bad

TABLE 3. Records provided by each system

Name	Type	Description
<sys>logrecord	stringin	DHS log record
<sys>TIME:intSimulate	bo	1 = Simulating, 0 = not Simulating

4.3.1 <sys>TIME:health

The sequence program sets this record on the basis of the error return from the Bancomm card and the time difference between the Bancomm and an NTP server.

N.B. earlier versions of the time library recommended implementing this as an mbbo record. This had to be changed at v5 of this document due to the change of health values from 0, 1 and 2 to 0, 100 and 200. It is convenient though not essential to set the following fields on this record

- HIHI 200.0
- HIGH 100.0
- HSV MINOR
- HHSV MAJOR

Note also that this record does not provide the health in a format needed by the OCS. The OCS requires health to be returned as the strings "GOOD", "WARNING" or "BAD" from a SIR record. It is the responsibility of the subsystem developer to provide this additional functionality.

4.3.2 <sys>TIME:intSimulate

This record provides a public interface to the simulation state of the time library software running in the subsystem. If this record is being included in the alarm tree for the subsystem, it is convenient though not essential to set the following fields on this record

- OSV MINOR
- ONAM True
- ZNAM False

4.3.3 <sys>:logrecord

It has always been intended that this should be implemented as a DHS log record. At present this is not available and so should simply be implemented as a stringin or stringout record. Note also that "TIME:" is not part of the record name as at the time of writing timelib there was to be a single DHS log record in each subsystem.

This area of timelib may have to change in future releases when the status of logging requirements is clarified.

4.4 C code

A full explanation of the different time scales employed by the TCS can be found in [3] together with details of the routines that can be called from user written C code. Brief

descriptions of each routine are reproduced in Section 5.0 as extracted from the source code.

4.5 Diagnostic facilities

Apart from the standard EPICS facilities, a number of diagnostic routines are provided that can be used to check the operation of the time system

4.5.1 timeClockReport

This routine performs the same checks as timeClockCheck (Section 5.2) but rather than then possibly modifying the time system setup to ensure a valid configuration it simply reports the status to the terminal or telnet session connected to the IOC. The output below is shown for an IOC that has been started in simulation mode

-> `timeClockReport`

Time system is in simulation mode
All times are being read from the Real Time Clock

The following output was obtained from a Bancomm 637 when it is locked to the GPS satellite

-> `timeClockReport`

Status of bancomm hardware

Number of GPS leap seconds 11

Status return from reading Bancomm time: 0

The number of GPS leap seconds is the difference between the UTC and GPS time scales. The possible values of status return are documented in [1]. In this case there are no errors being reported by the Bancomm hardware.

4.5.2 timeDump

This routine simply dumps the contents of the time libraries internal variables to the screen. The output below was obtained on an IOC that was running the time system in simulation mode

-> `timeDump`

Dump of contents of time Library private memory

Init flag = 1

Offset bias (days) = 0.000

TT - TAI = 32.184

MJD of next leap sec = 999999.900

TAI - UTC before that date (secs) = 30.000

UT1 - UTC before that date (secs) = 0.746

Current TDB - TT (days) = 0.0000000000

Current LAST - GMST (rads) = 3.5697067474

If the same command is given on an IOC that has a Bancomm 637 card and is locked to a GPS satellite then the following output is obtained.

-> `timeDump`

Dump of contents of time Library private memory

Init flag = 1

Offset bias (days) = 0.000

TT - TAI = 32.184

MJD of next leap sec = 999999.900

TAI - UTC before that date (secs) = 30.000

UT1 - UTC before that date (secs) = 0.746

Current TDB - TT (days) = -0.0000000180

Current LAST - GMST (rads) = 3.5697067513

The interpretation of these values is as follows

- Init flag - a value of 1 confirms that the time library has been successfully initialised i.e. that timeInit has been called.
- Offset bias - this will normally be zero. It will only be non-zero if the time system is being run at a time offset from the present (see Section 3.6 on page 6)
- TT - TAI - the difference in seconds between TT and TAI
- MJD of next leap second - the value shown here is the default used prior to the announcement of when the next leap second will occur. Once a date has been announced the MJD will appear here.
- TAI - UTC - this is the number of leap seconds used to convert between TAI and UTC
- UT1 - UTC - the current difference between UT1 and UTC
- TDB - TT - the difference between TDB and TT. The output for the system simulating time is 0.0 which is the default.
- LAST - GMST - the current difference between LAST and GMST

4.5.3 timeCheck

This routine only returns useful information if the time system is *not* being simulated and the Bancomm card is in an error state. Under these conditions it will report back the difference in time between an NTP server and the Bancomm card. Further details are given in Section 5.1 . An example of its use is shown below

```
-> x = 1000.0
x = 0xd54fec: value = 1000
-> timeCheck(&x)
value = 1 = 0x1
-> x
x = 0xd54fec: value = 0.004
```

In this case the Bancomm is reporting that it is unsynched (since it returned 1) and that there is 4ms difference between it and the NTP server.

5.0 Reference Section

A number of calls within the time library make use of an enumerated type “timescale”. The following values are available:

TAI	International Atomic Time
UTC	Coordinated Universal Time
UT1	Universal Time
TT	Terrestrial Time
TDB	Barycentric Dynamical Time
GMST	Greenwich Mean Sidereal Time
LAST	Local Apparent Sidereal Time

5.1 timeCheck - Compare and check time standards

Purpose: Compare and check time standards

Description: This routine will check if the time returned by the Bancomm card is consistent with that fetched by NTP. This is only done if the time system isn't being simulated and reading the Bancomm returns an error. The routine is needed to try and separate serious errors reported by the Bancomm from less serious ones. For example, the Bancomm 637 will report errors whilst it is locking to the satellite signals, this can take 10s of minutes during which time a perfectly satisfactory time signal is being generated

Invocation :

timeCheck(&diff)

Parameters : (">" input, "!" modified, "<" output)

< timeDiff double Difference between BC and NTP (secs)

Function value :

< status int Status from bc635_read. Possible values are

bit0	time is unsynched
bit1	time offset too large
bit2	Frequency offset error

External functions :

bc635_read	Bancomm library	Reads raw TAI
TSgetUnixTime	EPICS Time Stamp library	Fetch UTC from NTP server

5.2 timeClockCheck - Checks the time initialisation parameters

Purpose : Checks the time initialisation parameters set by timeClockInit are consistent with the hardware that is present.

Description : The routine does a number of consistency checks to verify that the parameters set in the call to timeClockInit are valid. For example the IOC may have been booted to read time from a Bancomm card but actually there is no hardware present to allow this.

Invocation :

timeClockCheck()

External functions :

bcTestCard	Bancomm library	Check if Bancomm card is present
bc635_read	Bancomm library	Read raw TAI
bcReadStat	Bancomm library	Read status of Bancomm hardware
bcGetGpsLeap	Bancomm library	Fetch number of GPS leap seconds

Prior requirements : The routine timeClockInit should have been called first

Deficiencies : The main problem is trying to identify whether a Bancomm 637 or 635 card is present. This has to be done by trying to fetch the number of GPS leap seconds. If the number comes back as 0 it is a 635, if the number is positive then it is a 637.

5.3 timeClockInit - Initialise the clock used by the Gemini time system

Purpose : Initialise the clock used by the Gemini time system This routine must be called prior to starting EPICS.

Description : This routine is essential to the correct startup of the Gemini time system. It must be called prior to the startup of EPICS since it calls BCconfigure. Since EPICS is not running when this routine is called it uses private memory to store whether the time system is to be simulated and whether this IOC is the time bus master. These stored values can be used later by EPICS code. The master flag is set to 1 if the IOC in which this code is running is to be the time bus master i.e. it has a bc637 card

installed. The simulate flag is set to 1 if time is to be simulated i.e time is to be driven by an onboard real time clock rather than a Bancomm card.

Invocation :

timeClockInit(master, simulate, intPerSecond, intPerTick, timeOffset)

Parameters : (">" input, "!" modified, "<" output)

>	master	int	1 if timebus master 0 otherwise
>	simulate	int	1 if time is simulated 0 therwise
>	intPerSecond	int	Frequency of interrupts (Hz)
>	intPerTick	int	No. of interrupts per clock tick
>	timeOffset	int	Compensation offset (microsec)

Function value :

<	status	int	0 = OK

External functions :

BCconfigure	Bancomm library	Configure Bancomm time card

5.4 timeClockReport - Report on status of time system

Purpose : Report on status of time system

Description : This routine does many of the checks that are done by timeClockCheck. It differs in being a purely diagnostic function i.e. it makes no attempt to correct any inconsistencies it finds

Invocation :

timeClockReport()

External functions :

bcTestCard	Bancomm library	Check if Bancomm card is present
bc635_read	Bancomm library	Read raw TAI
bcReadStat	Bancomm library	Fetch bancomm hardware status
bcGetGpsLeap	Bancomm library	Fetch number of GPS leap seconds

Prior requirements : The routine timeClockInit should have been called first

Deficiencies : The main problem is trying to identify whether a Bancomm 637 or 635 card is present. This has to be done by trying to fetch the number of GPS leap seconds. If the number comes back as 0 it is a 635, if the number is positive then it is a 637.

5.5 timeDump - Outputs time library internal variables to screen

Purpose : Outputs time library internal variables to screen

Description : This is a purely diagnostic function used to print out the contents the the time library's internal variables to the screen.

Invocation :

```
timeDump()
```

5.6 timeGetMasterIOC - Get flag that indicates if this is the timebus master

Purpose : Get flag that indicates if this is the timebus master

Description : Fetch the masterIOC flag from the time library global memory

Invocation :

```
timeGetMasterIOC (&master)
```

Parameters : (">" input, "!" modified, "<" output)

<	master	int *	returned masterIOC flag

External variables :

>	masterIOC	int	The flag that says if this is the master IOC

Prior requirements : timeSetMasterIOC should have been called prior to this function

5.7 timeGetSimFlag - Fetch time simulation flag

Purpose : Fetch time simulation flag

Description : This routine fetches the simulation flag from the time library global memory . The flag says whether the subsystem is being simulated internally by the TCS.

Invocation :

timeGetSimFlag (&simulate)

Parameters : (">" input, "<" output, "!" modified)

<	simulate	int	simulate flag

External variables :

>	tsim	int	simulation flag

5.8 timeInit - Initialize (or re-initialize) the Gemini time library.

Purpose : Initialize (or re-initialize) the Gemini time library.

Description : This routine initialises the Gemini time library. If it is not called explicitly it will be called automatically when one of the other time library routines are called. The routine performs a number of consistency checks to ensure that the correct hardware is present to support the mode in which the time library is being run.

Invocation :

timeInit()

Function value :

<	status	int	Return status, 0 = OK, else = error

External variables :

<	initd	int	Initialised flag
>	elongt	double	True longitude of site (radians)
<	delstr	double	current LAST - GMST (radians)

5.9 timeLibRefresh - Updates slowly-changing variables in timeLib internal memory

Purpose : Updates slowly-changing variables in timeLib internal memory

Description : Store new values of TDB-TT and LAST-GMST in internal memory

Invocation :

timeLibRefresh ( )

Parameters : (">" input, "!" modified, "<" output)

>	deltdb	double	current TDB-TT (days)

External variables :

>	elongt	double	east longitude (true, radians)
<	deltdbd	double	current TDB-TT (days)
<	delstr	double	current LAST/GMST (radians)

5.10 timeNow - Read the raw time (including any offset-for-testing)

Purpose : Read the raw time (including any offset-for-testing)

Description : If running on a vxWorks system, read the current raw time from the Gemini Time Service. If running off line on a Unix system then read the Unix clock. In this case, the highest resolution clock available is used (this may only give a resultion of 1 second) In both cases the raw time returned is in SI seconds since 1970 January 1.0 TAI.

Invocation :

status = timeNow (&rawt)

Parameters : (">" input, "!" modified, "<" output)

<	rawt	double	raw time (SI seconds since 1970 Jan 1.0 TAI)

Function value :

<	status	int	0 = OK

External functions :

bc635_read	Bancomm library	Read raw time
clock_gettime	clockLib	Fetch current time as a timespec structure

External variables :

>	biass	double	offset for testing (sec): added to clock reading

5.11 timeNowC - Read the date and time

Purpose : Read the date and time

Description : Reads the time from the Gemini Time Service and expresses it in a specifed timescale as a calendar date and time.

Invocation :

status = timeNowC(scale, ndp, ymdhmsf)

Parameters : (">" input, "!" modified, "<" output)

>	scale	timescale	Enumerated type specifying timescale
>	ndp	int	number of digits in fraction
<	ymdhmsf	int[7]	year, month, day, hour, min, sec, fraction

Function value :

<	status	int	return status, 0 = OK

5.12 timeNowD - Read the Modified Julian Date

Purpose : Read the Modified Julian Date

Description : Read the time from the Gemini Time Service and express it in a specified timescale as a Modified Julian Date.

Invocation :

status = timeNowD (scale, &datemj)

Parameters : (">" input, "!" modified, "<" output)

>	scale	timescale	Time scale as enumeratd type
<	datemj	double *	Modified Julain Date (JD - 2400000.5)

Function value :

<	status	int	Return status, o = OK

5.13 timeNowR - Read the Sidereal time (in radians)

Purpose : Read the Sidereal time (in radians)

Description : Read the time from the Gemini time service and express it in a specified sidereal timescale as an angle in radians.

Invocation :

status = timeNowR (scale, &theta)

Parameters : (">" input, "!" modified, "<" output)

>	scale	timescale	Timescale as an enumerated type
<	theta	double *	Angle in radians

Function value :

<	status	int	Return status, 0 = OK

5.14 timeNowT - Read time of day

Purpose : Read time of day

Description : Read the time from the Gemini time service and express it in a specifed timescale as a time of day.

Invocation :

status = timeNowT (scale, ndp, hmsf)

Parameters : (">" input, "!" modified, "<" output)

>	scale	timescale	Time scale as an enumerated type
>	ndp	int	Number of digits required in fraction
<	hmsf	int[4]	hour, min, sec, fraction

Function value :

<	status	int	Return status, 0 = OK

5.15 timeOffline - Simulate Gemini TCS time service

Purpose : Simulate Gemini TCS time service

Description : Simulate the Gemini TCS time service in order to be able to run the time library functions offline.

Invocation :

status = timeOffline (tai, etlong, phi, hm, dleap, dat, dut)

Parameters : (">" input, "!" modified, "<" output)

>	tai	double	TAI (MJD)
>	elong	double	east longitude (true, radians)
>	phi	double	latitude (true geodetic, radians)
>	hm	double	height above reference spheroid (metres)
>	dleap	double	MJD following the latest known leap second
>	dat	double	TAI-UTC before that date (seconds)
>	dut	double	UT1-UTC before that date (seconds)

Function value :

<	status	int	Return status, 0 = OK

5.16 timeOffset - Set the offset-for-testing

Purpose : Set the offset-for-testing

Description : Set the offset-for-testing, which the timeNow function will then add to the clock reading. The purpose of this function is to allow running the observatory on a time offset from the present for testing purposes.

Invocation :

```
status = timeOffset(offset)
```

Parameters : (">" input, "!" modified, "<" output)

>	offset	double	Offset (in days)

Function value :

<	status	int	Return status, 0 = OK

5.17 timePrint - Prints the current time (according to timelib) on standard output

Purpose : Prints the current time (according to timelib) on standard output

Description : This is a purely diagnostic function used to print out the current time in order to check that an IOC has the correct time.

Invocation :

timePrint(scale)

Parameters : (">" input, "!" modified, "<" output)

>	scale	char *	"UTC","UT1","TT","TDB","GMST","LAST"

5.18 timeRaw2tai - Convert a raw time to TAI

Purpose : Convert a raw time to TAI

Description : Convert a raw time to TAI

Invocation :

tai = timeRaw2tai (raw)

Parameters : (">" input, "!" modified, "<" output)

>	raw	double	Raw time

Function value :

<	tai	double	TAI

5.19 timeSetDefaults - Set defaults in the time library private memory

Purpose : Set defaults in the time library private memory

Description : Simply copy data passed in as parameters into local memory after performing any unit conversions.

Invocation :

timeSetDefaults(tlongm, tlatm, xpmr, ypmr, ttmtai, mjdls, delat, delut, deltdb, bias)

Parameters : (">" input, "!" modified, "<" output)

>	tlongm	double	Site mean longitude (degrees)
>	tlatm	double	Site mena latitude (degrees)
>	xpmr	double	X component of polar motion (arcsec)
>	ypmr	double	Y component of polar motion (arcsec)
>	ttmtai	double	TT - TAI (secs)
>	mjdls	double	MJD following next leap second
>	delat	double	TAI - UTC before that date (secs)

>	delut	double	UT1 - UTC before that date (secs)
>	deltdb	double	current TDB - TT (secs)
>	bias	double	offset for testing (secs)

External functions :

slaPolmo	slalib	Compute components of polar motion

5.20 timeSetMasterIOC - Set flag to show if this is the timebus master

Purpose : Set flag to show if this is the timebus master

Description : Set the master IOC flag in the time library global memory

Invocation :

timeSetMasterIOC (master)

Parameters : (">" input, "!" modified, ">" output)

>	master	int	1 = masterIOC

External variables :

>	masterIOC	int	Flag to indicate if this is a master IOC

5.21 timeSetSimFlag - Set the time simulation flag

Purpose : Set the time simulation flag

Description : This routine sets the simulation flag in the time library global memory. The flag says whether the subsystem is being simulated internally by the TCS.

Invocation :

timeSetSimFlag (simulate)

Parameters : (">" input, "<" output, "!" modified)

>	simulate	int	simulate flag

External variables :

>	tsim	int	Time simulation flag

5.22 timeTai2raw - Convert a TAI int a raw time

Purpose : Convert a TAI int a raw time

Description : Convert a TAI int a raw time

Invocation :

raw = timeTai2raw (tai)

Parameters : (">" input, "!" modified, "<" output)

>	tai	double	TAI

Function value :

<	raw	double	Raw time

5.23 timeThenC - Convert a raw time into a date and time

Purpose : Convert a raw time into a date and time

Description : Express a previously read raw time as a claendar date and time in a specified timescale.

Invocation :

status = timeThenC (rawt, scale, ndp, ymdhmsf)

Parameters : (">" input, "!" modified, "<" output)

>	rawt	double	Raw time, SI seconds since TAI 1970 Jan. 1.0
>	scale	timescale	Timescale as an enumerated type
>	ndp	int	No. of digits in fraction
<	ymdhmsf	int[7]	Year, month, day, hour, min, sec, fraction

Function value :

<	status	int	Return status, 0 = OK

5.24 timeThenD - Convert a raw time into a Modified Julian Date.

Purpose : Convert a raw time into a Modified Julian Date.

Description : Expresses a previously read raw time as a Modified Julian Date in a specified timescale.

Invocation :

status = timeThenD (rawt, scale, &datemj)

Parameters : (">" input, "!" modified, "<" output)

>	rawt	double	raw time, SI seconds since TAI 1970 Jan 1.0
>	scale	timescale	Time scale as an enumerated type
<	datemj	double *	Modified Julian date (JD - 2400000.5)

Function value :

<	status	int	Return status, 0 = OK

5.25 timeThenR - Converts a raw time into a sidereal time

Purpose : Converts a raw time into a sidereal time

Description : Express a previously read raw time as a sidereal time (in radians) in a specified timescale.

Invocation :

status = timeThenR (rawt, scale, &theta)

Parameters : (">" input, "!" modified, "<" output)

>	rawt	double	raw time, SI seconds since TAI 1970 Jan. 1.0
>	scale	timescale	Time scale as an enumerated type
<	theta	double *	Sidereal time (in radians)

Function value :

<	status	int	Return status, 0 = OK

5.26 timeThenT - Converts a raw time into a time of day.

Purpose : Converts a raw time into a time of day.

Description : Expresses a previously read raw time as a time of day in a specified timescale.

Invocation :

status = timeThenT (rawt, scale, ndp, hmsf)

Parameters : (">" input, "!" modified, "<" output)

>	rawt	double	raw time, SI seconds since TAI 1970 Jan. 1.0
>	scale	timescale	Time scale as an enumerated type
>	ndp	int	number of digits in fraction
<	hmsf	int[4]	hour, min, sec, fraction

Function value :

<	status	int	Return status, 0 = OK